Optimal Slack Stealing Servicing for Real-Time Energy Harvesting Systems

Article *in* The Computer Journal · July 2020

DOI: 10.1093/comjnl/bxaa047

3 authors:

Rola el Osta
University of Nantes
12 PUBLICATIONS 12 CITATIONS
SEE PROFILE

Maryline Chetto
University of Nantes
122 PUBLICATIONS 891 CITATIONS
SEE PROFILE

Hussein El Ghor
Lebanese University
34 PUBLICATIONS 121 CITATIONS
SEE PROFILE

Advance Access publication on
doi: 10.1093/comjnl/bxaa047

Optimal Slack Stealing Servicing for Real-Time Energy Harvesting Systems

Rola El Osta[1], Maryline Chetto[1] and Hussein El Ghor[2,*]

[1]*University of Nantes, LS2N Laboratory, Nantes, France*
[2]*Lebanese University, LENS Laboratory, Saida, Lebanon*
**Corresponding author: husseinelghor@hotmail.com*

We consider the problem of real-time scheduling in uniprocessor devices powered by energy harvesters. In particular, we focus on mixed sets of tasks with time and energy constraints: hard deadline periodic tasks and soft aperiodic tasks without deadlines. We present an optimal aperiodic servicing algorithm that minimizes the response times of aperiodic tasks without compromising the schedulability of hard deadline periodic tasks. The server, called Slack Stealing with energy Preserving (SSP), is designed based on a slack stealing mechanism that profits whenever possible from available spare processing time and energy. We analytically establish the optimality of SSP. Our simulation results validate our theoretical analysis.

Keywords: real time; energy harvesting; scheduling; earliest deadline first; slack stealing; aperiodic servicing

Received 10 April 2019; Revised 16 January 2020; Accepted 28 March 2020
Handling editor: Antonio Fernandez Anta

1. INTRODUCTION

The lifetime of battery-powered embedded devices is limited by the amount of energy that can be stored in the batteries. Furthermore, in many applications such as submarine, nuclear and medical, the location of these devices renders the activity of replacing batteries infeasible or very costly. As a consequence, green solutions based on ambient energy have appeared in the past decade. Intensive research has been conducted on energy harvesting (EH) for small devices such as wireless sensors. Solar, vibrational and thermal energy among others can be scavenged from the surroundings of an embedded device to replenish its battery or capacitor. The technical challenges to achieve energy autonomy and to make EH systems work effectively were initially described in [1], [2]. Among these new challenges of researchers, EH aware scheduling is a very important one.

An EH system consists of a single processing unit with unique voltage and frequency, a set of jobs, an energy storage unit, an EH unit and an energy source (Fig. 1).

In this paper, we focus on hard real-time systems that require *a priori* guarantee that all timing constraints can be met. Real-time schedulability theory provides firm guarantees for periodic executions of a set of tasks on uniprocessors in terms of assurances to meet deadlines (see [3] and [4] for surveys). The schedulability theory for real-time systems relies on *a priori* knowledge of the worst-case execution time (WCET) of hard real-time tasks to check if the deadline of a task can be met. A safe upper bound on the WCET of a task can be provided through static analysis, dynamic analysis or even a combination of both techniques.

Conventional task scheduling techniques are non-idling, i.e. make the processor necessarily busy if at least one task is pending for execution. With dynamic priority assignment, task priorities are assigned to individual jobs. For example, earliest deadline first (EDF) assigns the highest priority to the task with the closest deadline [5]. EDF is optimal and achieves 100% processor utilization with no energy consideration [6]. Recently, it has been pointed out that EDF is the best non-idling strategy for systems with EH capabilities although it may behave very poorly [7]. We have shown a new idling scheduling algorithm, namely energy deadline with energy harvesting (ED-H) to be optimal: if a task set can be scheduled by any algorithm on a platform composed of given processor, energy harvester and energy reservoir, then it can be scheduled using the ED-H algorithm on the same platform [8]. ED-H dynamically assigns priorities based upon the deadlines of the tasks as EDF but provides an optimal solution at the cost of clairvoyance features since its relies on accurate prediction on the incoming energy source.

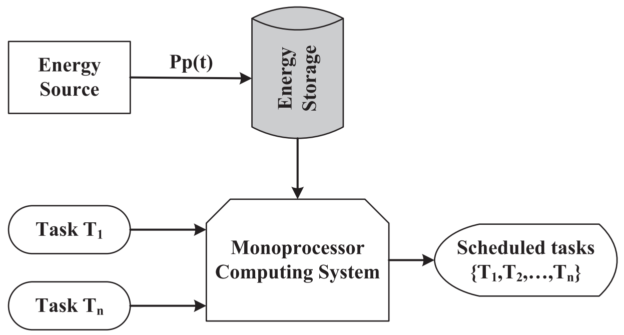
Energy Source
Pp(t)
Energy Storage
Task T$_1$
Task T$_n$
Monoprocessor Computing System
Scheduled tasks {T$_1$,T$_2$,…,T$_n$}

FIGURE 1. A real-time energy harvesting system.

Real-time systems must be able to handle not only periodic tasks but also soft aperiodic tasks with irregular arrival times and no deadlines. Periodic tasks are generally used to implement activities such as sensory acquisition or control loops, which need to be executed at constant rates to insure system stability. Hence, periodic tasks often have hard deadlines that must be met under all anticipated circumstances. On the other hand, aperiodic tasks are usually employed to implement non-critical activities. The goal of the scheduler is to guarantee the deadlines of periodic task while providing good response times for soft aperiodic tasks.

There have been extensive studies on the scheduling of periodic tasks and soft aperiodic tasks for several decades [9]. Fixed priority as well as dynamic priority algorithms can be found in the literature [4]. For example, Spuri and Buttazzo proposed the total bandwidth (TB) server that provides optimal responsiveness for dynamic priority systems [10]. Under the TB server, a suitable deadline is assigned to each occurring aperiodic task so as to schedule it according to the EDF algorithm together with the periodic tasks.

Chetto H. and Chetto M. used a slack stealing technique to propose the so called earliest deadline as late as possible (EDL) server [11] [12] [13]. Slack stealing attempts to make time for servicing aperiodic task by stealing all the processing time it can from the periodic tasks [14][15].

The above approaches target a single processor and independent task sets, in which no energy constraint is defined.

This paper presents several contributions:

- An aperiodic task servicing algorithm, called Slack Stealing with energy Preserving (SSP), is provided for dynamic priority systems with EH constraints. SSP is an energy aware version of the EDL-based slack stealing server. In SSP, both energy consumed by the tasks and energy drawn by the harvester from the environment are considered. In addition to the system slack time, the so-called system slack energy is dynamically computed. This allows to determine when to execute any occurring aperiodic task with optimal responsiveness while guaranteeing no deadline violation for the periodic tasks and no energy starvation.
- We describe the optimality analysis of this novel aperiodic task server.
- This work integrates experimental simulations on randomly generated task sets to validate the proposed approach and provide quantitative results.

The rest of this paper is organized as follows: the system model is presented in the subsequent section. Section 3 gives background materials. The algorithm for soft aperiodic task handling and the proof of optimality are presented in section 4. Simulation results in section 5 illustrate its effectiveness. Section 6 includes the related work. Finally, section 7 concludes the paper.

2. ASSUMPTIONS AND TERMINOLOGY

All properties of the proposed algorithm will be proved under the following assumptions.

2.1. Task assumptions

We will consider the following assumptions: a periodic task set τ can be denoted as follows: $\tau = \{\tau_i \mid 1 \le i \le n\}$. Each periodic task τ_i has a period T_i, a relative deadline D_i, a constant WCET C_i (normalized to processor computing capacity) and a constant worst case energy requirement E_i. C_i and E_i can be derived by a static analysis of the source code. So we use a periodic task model with the four-tuple (C_i, E_i, D_i, T_i) associated with τ_i. We consider a constrained-deadline task set τ in which $0 < C_i \le D_i \le T_i$. Task τ_i generates jobs that are released at times 0, T_i, $2T_i$,... and must complete by times D_i, $T_i + D_i$, $2T_i + D_i$,... The hyperperiod H of a periodic task set is defined as the least common multiple (LCM) of the request periods T_i, that is $H = LCM(T_1, T_2, ..., T_n)$. The processor utilization of the periodic task set τ is $U_{pp} = \sum_{\tau_i \epsilon \tau} \frac{C_i}{T_i}$, which is less than or equal to 1. Similarly, we define the energy utilization of τ as $U_{ep} = \sum_{\tau_i \epsilon \tau} \frac{E_i}{T_i}$, which characterizes the average energy consumption of τ per time unit.

A job is any request that a task makes. A four-tuple (r_j, C_j, E_j, d_j) is associated with a job J_j and gives its release time, WCET, worst-case energy consumption and (absolute) deadline, respectively. The task set τ gives rise to an infinite set of jobs that are scheduled by the optimal uniprocessor scheduler ED-H. Throughout our discussion, we will assume that the processor has one operating frequency and its energy consumption is only due to dynamic switching energy.

Additional tasks called aperiodic arrive in the system irregularly. Each aperiodic task has WCET and worst-case energy requirement considered to be known at its arrival time i.e. the time at which the task is activated and becomes ready to execute. All aperiodic tasks are without deadlines and unpredictable arrival times. We will use the following notation throughout the paper: Ap is a stream of aperiodic occurrences defined as $Ap = Ap_i(a_i, c_i, e_i), i = 1..m$, where a_i is the arrival time, c_i is the WCET and e_i is the worst case energy requirement. The finish time of Ap_i will be denoted by f_i.

The overhead due to context (processor) switching and interrupt attention is assumed to be negligible compared with computational time of the tasks or, otherwise, included in it. Tasks do not suspend themselves or synchronize with other tasks.

2.2. Energy assumptions

At every time t, the harvester (e.g. solar panel) draws energy from ambient and converts it into electrical power with instantaneous charging rate $P_p(t)$ that incorporates all losses. The energy harvested in the time interval $[t_1, t_2)$ is thus given by $E_p(t_1, t_2) = \int_{t_1}^{t_2} P_p(t)\mathrm{d}t$. We assume that the energy production times can overlap with the consumption times. The energy consumed in any unit time slot is no less than the energy produced in the same unit timeslot. Consequently, the residual capacity of the energy storage is never increasing every time a job executes. And the remaining energy needed by any job executing on the processor has to be drawn from the reservoir. The energy produced by the source is not controllable and not necessarily a constant value.

But we can predict it accurately for the near future with negligible time and energy cost.

Our system uses an ideal energy reservoir (e.g. supercapacitor or rechargeable battery) to continue operation even when there is no energy to harvest. Its nominal capacity C corresponds to the maximum amount of energy that can be stored at any time. The energy reservoir receives power from the harvester and delivers power to the processor. The stored energy at any time t is denoted $E(t)$. The energy reservoir does not leak any energy over time. If it is fully charged at time t and we continue to charge it, energy is wasted. In contrast, if it is fully discharged at time t (energy depletion), no job can be executed.

2.3. Origins of deadline violation

According to the previous model, a job misses its deadline if one of the two following situations occurs: when the job reaches its deadline at time t, its execution is incomplete because the time required to process the job by its deadline is not sufficient. When the job reaches its deadline at time t, its execution is incomplete because the energy required to process the job by its deadline is not available. The energy in the reservoir is exhausted when the deadline violation occurs.

In subsequent sections, we will assume that no deadline violation may occur. In other terms, the periodic task set is schedulable using the optimal deadline driven scheduler ED-H.

TABLE 1. Parameters of periodic tasks.

Task	C_i	D_i	T_i
τ_1	2	8	9
τ_2	2	10	12
τ_3	2	15	18

3. PREVIOUS WORK

3.1. Aperiodic servicing

Many real-time systems contain non-periodic tasks that have significantly varying arrival times and no deadline. Scheduling algorithms must be able to provide good average response times for aperiodic tasks without jeopardizing the schedulability of the periodic ones, even though their arrivals are non-deterministic.

One common approach is background processing: aperiodic requests are served only whenever the processor is idle i.e. no periodic tasks are ready to run. This kind of aperiodic scheduling is the simplest possible. However, if the load of the periodic task set is high, then utilization left for background service is low, and service opportunities are relatively infrequent.

Other algorithms have been proposed in the literature [4]. The slack stealer is one of them, which has been specifically designed for EDF.

The slack stealing is a class of algorithms that allocate dynamically idle processor capacity to non-periodic tasks. This approach overcomes the drawbacks of background servicing. It is optimal in the sense that it minimizes the response times of aperiodic tasks among all algorithms while meeting the hard deadlines of the periodic tasks [11]. Moreover, it can handle burstly arrivals of aperiodic requests and does not require to know *a priori* their WCETs.

The idea is to postpone the execution of periodic activities, making any spare processing time available as soon as possible for the aperiodic activities. The slack time is computed dynamically whenever necessary, using a preprocessing step for more efficiency. When there are no aperiodic activities in the system, the periodic tasks are scheduled according to the EDF algorithm. A means of determining the slack time is thus key to the operation of the algorithm (see [13] for details). The worst-case complexity of the algorithm is O(m.n) where $m = O(T_n/T_1)$ and n is the number of periodic tasks.

3.1.1. *Using the slack: an example*

We show how slack in EDL server can be used to execute soft aperiodic tasks. Let us consider a real-time system with three periodic tasks whose real-time parameters are shown on Table 1.

Figure 2 shows how tasks are scheduled from time 0 to H. Arrows represent the time when task is invoked. We show five idle intervals at times 6, 11, 14, 22, 26 and 29, respectively.

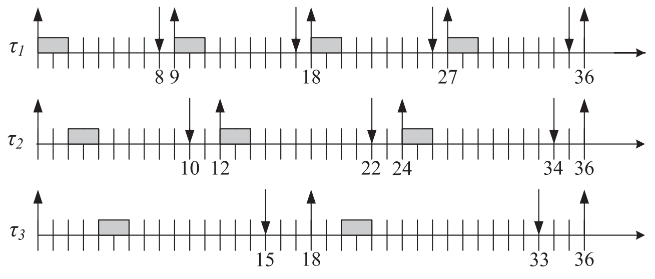

FIGURE 2. Tasks scheduled by the EDF discipline.

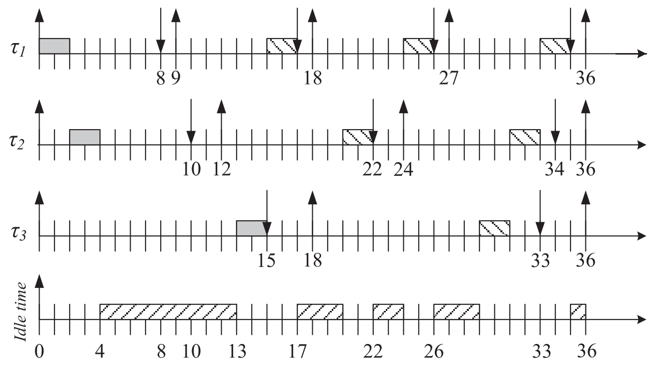

FIGURE 3. Tasks delayed during maximum amount of time through the slack stealing mechanism.

Let us assume an aperiodic task arrives at time 4 (Fig. 3). Postponing the execution of the periodic tasks as much as possible from time 4, we can produce idle intervals at times 4, 17, 22, 26 and 35, respectively. When hard real-time tasks are delayed the maximum amount of time that they can support, the response time of soft tasks is minimized. If no soft task is ready for execution, then a periodic task is executed and the slack remains available for the future.

In the next section, we will extend the slack stealer server to deal with energy constraints.

3.2. ED-H scheduling

In a system with limitations and fluctuations in energy availability, simply executing jobs according to the EDF rule, either as soon as possible (EDS) or as late as possible (EDL) may lead to violate some deadlines because of energy starvations. This is why, in energy constrained systems, dynamic power management plays a crucial role due to its impact on the resulting performance. The dynamic power management rule permits to decide when to put the processor in the active mode and for how long time. The objective of such a policy is to prevent from energy depletion while still preserving the system from deadline violation. Consequently, we presented a novel energy-aware scheduling algorithm ED-H (earliest deadline under energy harvesting settings) and we proved it to be optimal [8]. ED-H orders the ready jobs according to the EDF rule but performs a test before dispatching the highest priority job so as to prevent from energy starvation. More precisely, if the decision test receives a ‘yes’ answer, the processor is authorized to be in the active mode since two conditions can be satisfied. Firstly, the energy level in the energy reservoir is sufficient enough to execute the active job during at least one time unit. Secondly, executing the active job will not provoke any energy starvation for a future occurring job. The decision test may lead to a ‘no’ answer. The processor has to sleep so that the energy storage unit recharges sufficiently. Deciding for how long time recharging should be performed is very flexible. And we may opt to stop recharging the energy reservoir only when the reservoir is replenished or the system has no more slack time.

ED-H needs to maintain the following dynamic data for every job: the slack time and the slack energy.

The slack time of a hard deadline job J_i at current time t is

$$ST_{J_i}(t) = d_i - t - h(t, d_i) \tag{1}$$

where $h(t, d_i)$ is the total processing demand of uncompleted jobs at t with deadline at or before d_i. $ST_{J_i}(t)$ gives the available processor time after executing uncompleted jobs with deadlines at or before d_i. We may then define the slack time of a periodic task set τ at current time t as follows:

$$ST_\tau(t) = \min_{d_i > t} ST_{J_i}(t). \tag{2}$$

The slack time as computed with (2) gives the maximum continuous processor time that could be made available from time t while still guaranteeing the deadlines of all the jobs generated by τ.

Similarly, we define the slack energy of J_i at current time t by equation 3.

$$SE_{J_i}(t) = E(t) + E_p(t, d_i) - g(t, d_i) \tag{3}$$

where $g(t, d_i)$ represents the total energy required by jobs on the time interval $[t, d_i)$. It concerns both jobs that are ready at t but not completed at d_i and future jobs, with deadline less than or equal to d_i.

Clearly, $SE_{J_i}(t)$ represents the maximum energy surplus that could be consumed within $[t, d_i)$ while guaranteeing enough energy for jobs with deadline less than or equal to d_i. In other words, if there exists some job J_i such that $SE_{J_i}(t) = 0$, executing any other job with a deadline higher than d_i within $[t, d_i)$ will involve energy starvation for J_i.

Hence, the slack energy of the periodic task set τ at current time t represents the maximum energy surplus that the system could consume instantaneously at t. The slack energy at t is

$$SE_\tau(t) = \min_{t < d_i} SE_{J_i}(t). \tag{4}$$

TABLE 2. Parameters of periodic tasks with energy considerations.

Task	C_i	E_i	D_i	T_i
τ_1	2	9	8	9
τ_2	2	8	10	12
τ_3	2	9	15	18

Finally, the preemption slack energy at the current time t gives the maximum energy that could be consumed by the active job while guaranteeing absence of energy starvation for jobs that may preempt it. Let d be the deadline of this active job. The preemption slack energy at t is

$$PSE(t) = \min_{t<d_i<d} SE_{J_i}(t). \tag{5}$$

Let us use the following notation:

- t: current time
- $L_r(t)$: list of periodic jobs ready to be processed
- $A_r(t)$: list of aperiodic jobs ready to be processed
- $E(t)$: residual capacity of the energy reservoir
- $ST(t)$: slack time of the periodic task set
- $SE(t)$: slack energy of the periodic task set
- $PSE(t)$: preemption slack energy of the periodic task set

The operations of the ED-H scheduler are as follows.

- **Rule 1:** The EDF priority order is used to select the future running job in $L_r(t)$.
- **Rule 2:** The processor is imperatively idle in $[t, t+1)$ if $L_r(t) = \emptyset$.
- **Rule 3:** The processor is imperatively idle in $[t, t+1)$ if $L_r(t) \neq \emptyset$ and either $E(t) = 0$ or $PSE(t) = 0$.
- **Rule 4:** The processor is imperatively busy in $[t, t+1)$ if $L_r(t) \neq \emptyset$ and either $E(t) = C$ or $ST(t) = 0$
- **Rule 5:** The processor can equally be idle or busy if $L_r(t) \neq \emptyset$, $0 < E(t) < C$, $ST(t) > 0$ and $PSE(t) > 0$.

The result of optimality of the ED-H scheduler is recalled in Theorem 3.1. If a task set is schedulable by any algorithm on a platform composed of given processor, energy harvester and energy reservoir, then it is schedulable using the ED-H algorithm on the same platform.

Theorem 3.1. *[8] The ED-H scheduling algorithm is optimal.*

The following example illustrates the benefits of ED-H under EDF. Let us consider a task set Γ of three periodic tasks as depicted in Table 2.

We assume that the energy storage capacity is $C = 7$ energy units at $t = 0$. For simplicity, we assume that the rechargeable power is constant along the hyperperiod and equal to 3 ($Pp = 3$).

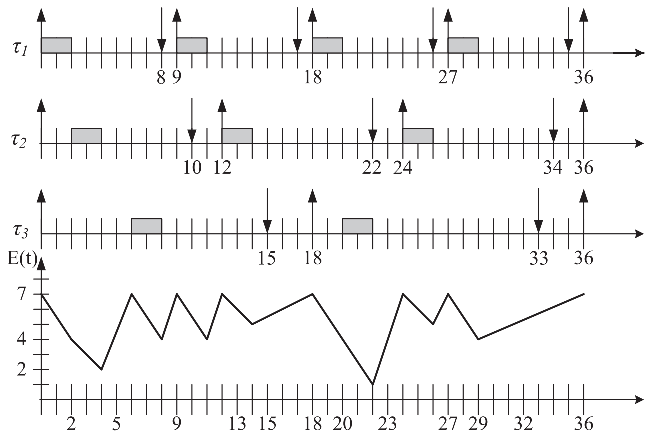

FIGURE 4. Tasks scheduled according to ED-H.

Let us schedule Γaccording to ED-H within the first hyperperiod. We verify that Γ is schedulable because all tasks are executed within their deadlines and the energy reservoir is full at the end of the hyperperiod (Fig. 4).

4. HARVESTING-AWARE SLACK STEALING

4.1. Slack analysis under ED-H

The main principle of the slack stealer for aperiodic servicing with ED-H is to authorize aperiodic job executions as long as this does not involve a deadline violation for all the jobs generated by τ. Let us recall that a deadline violation occurs either because of processing time starvation or energy starvation. This leads us to consider the system slack at time t as a pair of values. The first one is the slack time of τ defined as the maximum time the system could be delayed for executing additional tasks from t. The second one is the slack energy of τ defined as the maximum energy surplus that the system could consume at t.

4.2. The slack stealing algorithm

We present here a new slack stealing algorithm that extends the original one to EH settings. The original slack stealer is greedy in that sense that the available slack time is always consumed if there is an aperiodic task ready to run. The basic idea of the proposed SSP is based on the slack time to execute the aperiodic tasks as soon as possible and the slack energy to avoid any energy starvation for periodic tasks. If no aperiodic task arrives, periodic tasks are operated with ED-H. And if any aperiodic task arrives, it uses the collected slack time and slack energy to service aperiodic tasks. The slack stealer can be viewed as a task that is ready for execution whenever the aperiodic queue is non-empty. This task is suspended when the queue is empty. The slack stealer receives the highest priority whenever there is slack i.e. both slack time and slack energy.

It receives the lowest priority whenever there is no slack. The slack stealer selects the aperiodic task in FIFO order.

The scheduling framework can be described by the following pseudo-code:

Algorithm 1 ED-H with the Slack Stealing server

```
while True do
    if $A_r(t)$ is not empty then
        compute $ST(t)$ and $SE(t)$
        if $ST(t) > 0$ and $SE(t) > 0$ then
            execute the slack stealer
        else
            execute the ED-H scheduler
        end if
    else
        execute the ED-H scheduler
    end if
end while
```

4.3. Optimality analysis

The property of optimality, that is, the minimization of the response times of the aperiodic requests, is stated in the following theorems.

Theorem 4.1. *All periodic tasks meet their deadlines when scheduled according to ED-H with the slack stealer for aperiodic servicing.*

Proof. We prove the theorem by contradiction. Suppose that a job, say J_1, issued from a periodic task misses its deadline at d_1. And d_1 is the first deadline that is missed in the schedule. This violation is due to one of the two following reasons: the time starvation case is when deadline d_1 is missed with the energy reservoir not exhausted at d_1. The energy starvation case is when the reservoir is exhausted at d_1 and J_1 is not completed. As the periodic task set is feasible, the deadline violation necessarily comes from the execution of aperiodic tasks. Let t_0 be the latest time instant before d_1 where an aperiodic task, say Ap_0 executes. By definition of the slack stealer, Ap_0 was authorized to execute within $[t_0-1, t_0)$ because $ST(t_0-1) > 0$ and $SE(t_0-1) > 0$. And Ap_0 stops execution at t_0, because either the system has no more slack time i.e. $ST(t_0) = 0$ or the system has no more slack energy i.e. $SE(t_0) = 0$. Let us examine the two cases. *Case 1:* $ST(t_0) = 0$ The slack time, $ST(t_0)$ as computed at t_0 with (2), gives the maximum processing time that could be made available from time t_0 while still guaranteeing the deadlines of all the jobs issued from the periodic tasks ready at or from time t_0. The condition $ST(t_0) = 0$ guarantees that if the jobs are executed from time t_0 according to the EDF rule, all periodic jobs can complete by deadlines even if one of these jobs completes exactly at deadline. This contradicts that d_1 is violated. *Case 2:* $SE(t_0) = 0$ The slack energy, $SE(t_0)$ as computed at t_0 with (4), gives the maximum energy surplus that the system could consume instantaneously at t_0 while preventing an energy starvation for all the jobs issued from the periodic tasks ready at or after time t_0. From t_0 to d_1, no energy is wasted (definition of ED-H) and all the jobs that execute within $[t_0, d_1)$ are periodic ones. Consequently, there is no energy starvation, which contradicts the deadline violation at d_1 with $E(d_1) = 0$. As an example, we consider a set of three periodic tasks that we studied in the previous section. Suppose that the first aperiodic job Ap_1 has computation time 1, has an energy consumption 4 energy units and is released at $t = 4$. Another aperiodic task with computation time 1, an energy consumption 4 energy units, is released at $t = 14$. At time 0, the residual capacity is maximum since the storage capacity is full. τ_1 is the highest priority task, runs and finishes at time 2 and consumes nine energy units. At time 2, the residual capacity is given by $E_{max} - E_1 + P_p * C_1 = 4$. Now, τ_2 has the highest priority. It executes completely until time 4 and consumes eight energy units. The residual capacity equals two energy units. At time 4, Ap_1 is released. System slack time and system slack energy are then computed at time t=6. Slack time is given by the minimum of the slack time of all periodic instances in the system and computed as follows: $ST(\tau_1, 4) = d_1 - t - h(t, d_1) = 4, ST(\tau_2, 4) = d_2 - t - h(t, d_2) = 6$ and $ST(\tau_3, 4) = d_3 - t - h(t, d_3) = 15 - 4 - 2 = 9$. So, the slack time $ST_\tau(4) = min(ST(\tau_1, 4), ST(\tau_2, 4), ST(\tau_3, 4)) = 4 > 0$. Slack energy is given by the minimum of the slack energy of all periodic instances in the system and computed as follows: $SE(\tau_1, 4) = E(t) + E_p(t, d_1) - g(t, d_1) = E(4) + \int_4^8 P_p dt = 14, SE(\tau_2, 4) = E(4) + \int_4^{10} P_p \mathrm{d}t = 20$ and $SE(\tau_3, 4) = E(4) + \int_4^{15} P_p \mathrm{d}t - E_3 = 26$. So, the slack energy $SE_\tau(4) = min(SE(\tau_1, 4), SE(\tau_2, 4), SE(\tau_3, 4)) = 14 > 0$. As the storage unit is not empty, the slack time is positive (equals 4) and the slack energy is positive (equals 14), J_1 is authorized to execute and to consume a maximum of four energy units. After its execution, the residual capacity falls at 1. We let the processor idle in order to recharge the battery. At time 7, the storage unit has fulfilled and the highest priority task τ_3 executes completely according ED-H at time 9 where the residual capacity equals 4. We continue to schedule the tasks till time 14 where Ap_2 is released. Here, we have to check again if we abide by the three conditions: (i) the reservoir is not empty (equals 2), (ii) the slack time is positive and equals to 10 and (iii) the system slack energy is greater than zero and equals 5. Consequently, Ap_2 is authorized to execute immediately. Again, we continue to schedule the periodic tasks according to ED-H till the end of the hyperperiod where the energy reservoir leads to seven energy units (Fig. 5). We note that the aperiodic tasks Ap_1 and Ap_2 are executed at the earliest by utilizing the released idle times and energy surplus collected with regards to periodic tasks, while periodic tasks are deferred further in time. In the absence of aperiodic tasks in the system, periodic tasks

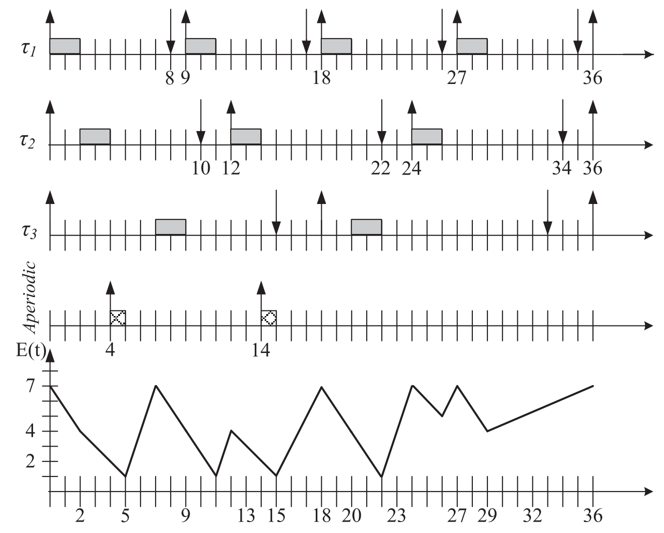

FIGURE 5. Tasks scheduled according to SSP with energy constraints.

are scheduled again under ED-H. In this case, the response times of aperiodic tasks Ap_1 and Ap_2 are one unit of time for both, which is a clear evidence of minimizing the aperiodic responsiveness. ■

Theorem 4.2. *For any periodic task set scheduled according to ED-H and a stream of aperiodic tasks processed in FIFO order, the slack stealing algorithm minimizes the response time of every aperiodic task, among all algorithms that are guaranteed to meet all deadlines.*

Proof. We prove the theorem by showing that any alternative algorithm, A, which results in a shorter response time for any aperiodic task, cannot guarantee that the deadlines of all the periodic tasks will be met. Let Ap_0 be the first aperiodic task that has a shorter response time when scheduled by algorithm A. As Ap_0 is the first such task, the response times of all previously serviced soft tasks must be the same as or longer than when scheduled by the slack stealer. Once at the head of the queue, Ap_0 is serviced by the dynamic slack stealer so long as $SE(t) > 0$ and $ST(t) > 0$. For algorithm A to result in a lower response time, it must process Ap_0 for at least one clock tick when the slack stealer is unable to do so. We denote the time at which this occurs by t_0. The slack stealer computes two data at time t_0. The first one is the slack time i.e. the spare processing time that may be stolen. The second one is the slack energy i.e. the spare energy that may be stolen. One of these two data is zero at time t_0. *First case*: $ST(t_0) = 0$. Hence, for at least one job of periodic task, say J_1, we have $ST_{J_1}(t_0) = 0$. In servicing aperiodic task Ap_0 from t_0 to $t_0 + 1$, algorithm A has therefore lead to $ST_{J_1}(t_0 + 1) = -1$ culminating in the impossibility to complete job J_1 by its deadline. Algorithm A cannot therefore guarantee that the deadline of job J_1 will be met. *Second case*: $SE(t_0) = 0$. Hence, for at least one job of periodic task, say J_1, we have $SE_{J_1}(t_0) = 0$. That means that any additional energy consumption between t_0 and $t_0 + 1$ leads to $SE_{J_1}(t_0 + 1) < 0$ culminating in insufficient energy to execute job J_1 entirely by its deadline. Algorithm A cannot therefore guarantee that the deadline of job J_1 will be met due to energy starvation. ■

5. EXPERIMENTAL RESULTS

In this section we will briefly discuss the results of a simulation study that was carried out to measure the performance of the slack stealer server. This evaluation includes two servers in addition to SSP. Background with Energy Surplus (BES) executes an aperiodic task only if no periodic task is pending for execution and the energy reservoir is fully replenished. Under Background with Energy Preserving (BEP), aperiodic tasks are executed if there is no awaiting periodic task and the system slack energy is positive so as to avoid energy starvation. BEP significantly enhances the performance of BES with additional overhead.

The total processing load U_p incorporates 50% of the periodic processor utilization U_{pp} and 50% of the aperiodic utilization U_{ps}. Identically, the total energy load U_e includes 50% of the periodic energy utilization U_{ep} and 50% of the aperiodic energy utilization U_{es}.

In the first set of experiments, performance of the servers is evaluated as a function of the energy load, for three processing loads. The second set of experiments is concerned with the performance trend of SSP server with respect to the reservoir size.

5.1. Simulation setup

The experiments include a task set composed of n = 20 periodic tasks generated with random periods, computation times and energy requirements. Periods and computation times are distributed uniformly in discrete time steps, depending on U_{pp}.

Energy consumption of every task is proportional to its period and depends on the setting of U_{ep}. Periodic task sets are assumed to be schedulable in terms of processing time and energy. Similarly, aperiodic tasks are generated according to desired values for U_{ps} and U_{es} by simulating a poisson aperiodic arrival.

In all experiments, each point in the curves is computed over 100 runs. The energy reservoir is initially full and the recharging power P_p is constant.

5.2. Relative performance under various time and energy conditions

Aperiodic responsiveness is measured for three processing load profiles: (i) weakly constrained with $U_p = 20\%$, (ii) fairly constrained with $U_p = 40\%$ and (iii) highly constrained with $U_p = 80\%$. U_e/P_p varies from 5% to 100% in order to show

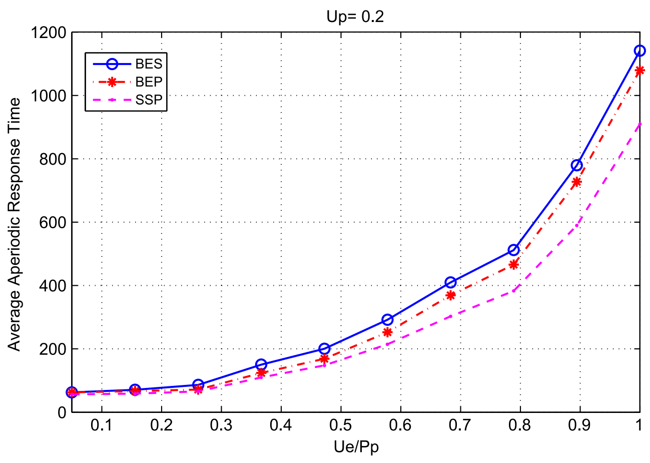

FIGURE 6. Average aperiodic response time with respect to U_e/P_p, for U_p=0.2.

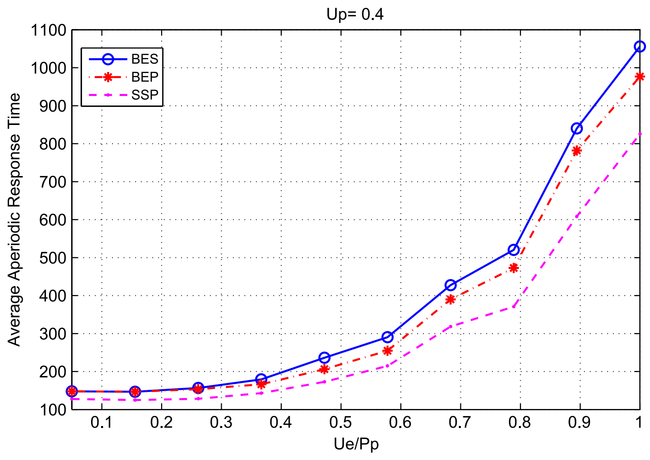

FIGURE 7. Average aperiodic response time with respect to U_e/P_p, for U_p=0.4.

the impact of energy availability on aperiodic responsiveness. Results are reported in Figs 6, 7 and 8, respectively.

As expected, the SSP server outperforms the two background policies BES and BEP for all configuration settings. It is worth mentioning that the higher the energy limitation, the wider the performance of SSP over the background techniques. BES shows inferior performance for high energy requirements since aperiodic tasks may execute only when the reservoir is fully replenished. BES and BEP behave similarly when renewable energy is greatly available in comparison to energy requirement.

For the first experiment, (Fig. 6, $U_p = 20\%$), we examine a system that is softly constrained by processor utilization. We can see that the slack stealer has aperiodic response time that is at least 25% lower compared to background servers for all energy conditions.

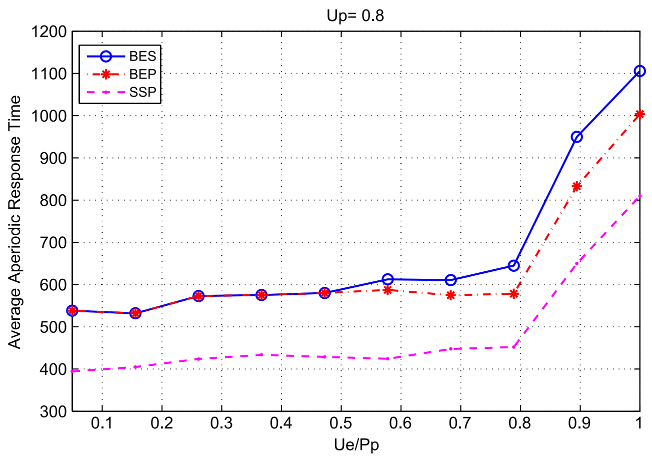

FIGURE 8. Average aperiodic response time with respect to U_e/P_p, for U_p=0.8.

TABLE 3. Relative performance with different reservoir sizes.

Reservoir capacity	U_e/P_p	Aperiodic response time		
		BES	*BEP*	*SSP*
E_{min}	0.2	2.4	2.1	1.7
	0.8	37.4	35.2	26.2
E_{min}	0.2	2.4	2.1	1.7
	0.8	37.4	35.2	26.2
$5 * E_{min}$	0.2	2.0	1.7	1.4
	0.8	23.0	15.8	14.7
$9 * E_{min}$	0.2	1.5	1.3	1.1
	0.8	13.4	8.7	6.3

For $U_p = 80\%$ (Fig. 8), the SSP server benefits from time slack stealing to optimize the processor utilization and performs much better than background servers. They both behave poorly even when there is no energy limitation. When the system is highly constrained both in terms of time and energy, the performance of the slack stealing-based server approaches that of the background servers.

5.3. Relative performance with different reservoir sizes

In this set of experiments, we evaluate the performance of the servers by varying the reservoir size with E_{min}, 5*E_{min} and 9*E_{min}. E_{min} is the minimum size of the reservoir that guarantees time and energy feasibility, for given U_p, U_e and P_p. Here, we report the results for systems that are no time constrained i.e. $U_p = 0.2$. In the Table 3, the third, fourth and fifth columns give the aperiodic responsiveness of BEP, BES and SSP servers, respectively, for two profiles in terms of energy constraints. Table 3 shows that the SSP server achieves significant reduction in aperiodic responsiveness, comparing with BEP and BES servers under all parameter settings. BEP achieves low aperiodic response time compared to BES. For example, when the system uses 20% of available energy with minimum reservoir size, the response time under SSP is 19% and 29% lower compared to BEP and BES, respectively. If the energy requirement is set to 80%, all servers record relatively high response times; however, the optimal slack stealer still beats the background servers by a large difference due to optimal exploitation of slack energy. For each of the three servers, higher is the size of the reservoir, lower is the normalized aperiodic response time for a given energy setting. If the reservoir size is set to E_{min} and the system uses 80% of available energy, the BES, BEP and SSP servers have aperiodic response time respectively equal to 37.4, 35.2 and 26.2. When increasing the reservoir size to $9 * E_{min}$, the response time of BES, BEP and SSP is respectively reduced by 64%, 75% and 76%. Such a significant improvement in aperiodic responsiveness comes from possible immediate service through extra energy that is available in the reservoir. We can see that the BES algorithm achieves the lowest reduction in response time over all the servers. This is because under BES, aperiodic job executions have to wait for the energy reservoir be fully replenished.

6. RELATED WORK

The critical challenges and opportunities to achieve energy-efficient communication in mobile cloud computing are described in [17], [18], [19] and [20]. This research focus on improving the energy efficiency and reducing data transmission overhead by storing the data required for computation in the cloud. Also, they present another approach by dynamically adjusting application partitioning between the cloud and mobile devices depending on the conditions of network.

A new method called eTime is presented by Shu *et al.* It is designed to prefetch frequently used data while deferring delay-tolerant data. It takes profit of the timing opportunity when network connectivity is good [19].

The method was evaluated in 2015 by proposing AppATP, an Application layer Adaptive Transmission Protocol targeting at energy-efficient data transfers between mobile devices and the cloud platform [21]. The same function of eTime is performed here based on measurements. Measurements show that less energy is consumed by mobile devices during good connectivity, and vice versa. Managing the use of renewable energy and the process of leveraging this energy in datacenters has gained attention. Research challenges in applying renewable energy in cloud computing datacenters are described from the following key aspects: generation models and prediction methods of renewable energy, capacity planning of green datacenters, intra-datacenter workload scheduling and load balancing across geographically distributed datacenters [22].

Renewable energy aware computing and online optimization algorithms have been highlighted in datacenters. Niu *et al.* [23] proposed an approach called JouleMR and integrate green-aware and cost-effective job/task scheduling into MapReduce. Real experiments and simulations show that JouleMR outperforms Greenhadoof [24] (up to 35% and 28% reduction, respectively) in terms of brown energy reduction. Furtheremore, JouleMR outperforms Greenhadoof (up to 30% and 36% reduction, respectively) in terms of electricity reduction.

Meanwhile, there are also some other works that focus on how to take advantage of datacenter in cloud computing. [24] provides a new method, fine-grained differential method (FGD), to evaluate the impact of power budget violation on latency-sensitive applications' performance. Wu *et al.* propose also, precise power capping (PPC), to improve power utilization and capacity of datacenters. Experimental results demonstrate that FGD and PPC behave more accurately when compared to other methods and strategies.

7. SUMMARY

Interest in EH has greatly increased over the past decade. EH technology permits to power small autonomous embedded devices without electric wires and overcomes the energy limitations of conventional battery-powered systems. In this paper, we addressed a real-time scheduling problem for a single processor device with EH capabilities.

We considered both hard deadline periodic tasks and aperiodic tasks with no deadline. Typically, aperiodic tasks benefit from being delivered as early as possible, while periodic tasks need to be guaranteed to meet their deadlines. Our contribution is an optimal aperiodic task server based on slack stealing and adapted to the optimal dynamic priority scheduler ED-H. The so-called SSP server schedules aperiodic tasks whenever the execution of periodic tasks may be safely postponed without causing missed deadlines. In other terms, SSP makes any spare processing time and any spare energy available as soon as possible. Through experiments, we have illustrated the improvements achieved by the SSP algorithm in aperiodic responsiveness compared to classical background servicing under different conditions. In the future, we will extend our current work to support Dynamic Voltage Frequency Scaling technology.

REFERENCES

[1] Kansal, A., Hsu, J., Zahedi, S. and Srivastava, M.B. (2007) Power management in energy harvesting sensor networks. *ACM T. Embed. Comput. S.*, 6, Article 32.

[2] Kansal, A. and Hsu, J. (2006) Harvesting Aware Power Management for Sensor Networks. *Proc. ACM/IEEE Design Automation Conference*, San Francisco, CA, July 24–28, pp. 651–656.

[3] Buttazzo, G. (2005) *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications*. Springer, Berlin.

[4] Liu, J. (2000) *Real-Time Systems*. Prentice Hall, United States.

[5] Dertouzos, M.-L. (1974) Control Robotics: The Procedural Control of Physical Processes. *Proc. Int. Federation for Information Processing Congress*, Stockholm, Sweden, August 5–10, pp. 807–813. North-Holland, Amsterdam and American Elsevier, New York.

[6] Liu, C.-L. and Layland, J.-W. (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20, 46–61.

[7] Liu, S., Lu, J., Wu, Q. and Qiu, Q. (2012) Harvesting-aware power management for real-time systems with renewable energy. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 20 1473–1486.

[8] Chetto, M. (2014) Optimal scheduling for real-time jobs in energy harvesting computing systems. *IEEE T. Emerg. Top. Com.*, 2, 122–133.

[9] Sprunt, B., Sha, L. and Lehoczky, J.P. (1989) Aperiodic task scheduling for hard real-time systems. *Real-Time Syst.*, 1, 27–60.

[10] Spuri, M. and Buttazzo, G. (1996) Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Syst.*, 10, 179–210.

[11] Chetto, H. and Chetto, M. (1989) Some results of the earliest deadline scheduling algorithm. *IEEE T. Software Eng.*, 15, 1261–1269.

[12] Spuri, M. and Buttazzo, G.C. (1994) Efficient Aperiodic Service Under Earliest Deadline Scheduling. *15th IEEE Real-Time Systems Symposium*, San Juan, Portorico, December, 2–11.

[13] Silly-Chetto, M. (1999) The EDL server for scheduling periodic and soft aperiodic tasks with resource constraints. *Real-Time Syst.*, 17, 1–25.

[14] Lehoczky, J.P. and Ramos-Thuel, S. (1992) An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed Priority Preemptive Systems. *Proc. IEEE Real-Time Systems Symposium, Phoenix, Arizona, USA, December 2–4, pp. 110–123*. IEEE Computer Society Press.

[15] Davis, R.I., Tindell, K.W. and Burns, A. (1993) Scheduling Slack Time in Fixed Priority Preemptive Systems. *Proc. RTSS 93, Raleigh Durham, North Carolina, USA, December 1–3, pp. 222–231*. IEEE Computer Society Press.

[16] Chetto, M. and Queudet, A. (2014) Clairvoyance and online scheduling in real-time energy harvesting systems. *Real-Time Syst.*, 50, 179–184.

[17] Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R. and Paramvir, B. (2010) MAUI: Making Smartphones Last Longer with Code Offload. *Proc. ACM MobiSys 10, San Francisco, CA, USA, June 15–18, pp. 49–62*. ACM Press, New York.

[18] Chun, B., Ihm, S., Maniatis, P., Naik, M. and Patti, A. (2011) Clonecloud: Elastic Execution Between Mobile Device and Cloud. *Proc. ACM EuroSys 11, Salzburg, Austria, April 10–13, pp. 301–314*. Association for Computing Machinery, New York, United States.

[19] Shu, P., Liu, F., Chen, M., Wen, F., Qu, Y. and Li, B. (2013) eTime: Energy-Efficient Transmission between Cloud and Mobile Devices. *Proc. IEEE INFOCOM, New York, April 14–19, pp. 195–199*. IEEE Press, Turin, Italy.

[20] Liu, F., SHU, P., Jin, H., Ding, L., Yu, J., Ni, D. and Li, B. (2013) Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wirel. Commun.*, 20, 14–22.

[21] Liu, F., Shu, P. and Lui, J. (2015) AppATP: an energy conserving adaptive mobile-cloud transmission protocol. *IEEE Trans. Comput.*, 64, 3051–3063.

[22] Deng, W., Liu, F., Jin, H., Li, B. and Li, D. (2014) Harnessing renewable energy in cloud datacenters: opportunities and challenges. *IEEE Netw.*, 28, 48–55.

[23] Niu, Z., He, B. and Liu, F. (2018) JouleMR: towards cost-effective and green-aware data processing frameworks. *IEEE Trans. Big Data*, 4, 258–272.

[24] Wu, S., Chen, Y., Wang, X., Jin, H., Liu, F., Chen, H. and Yan, C. (2018) Precise power capping for latency-sensitive applications in datacenter. *IEEE Trans. Sustain. Comput.*, 1, 1.